Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miha Polanc

Koliko točk določa katero geometrijsko ploskev?

diplomsko delo

visokošolski strokovni študijski program
računalništvo in informatika

Mentor: izr. prof. dr. Gašper Fijavž

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Prostorski geometrijski objekt lahko približno opišemo z vzorčenjem točk z njegove površine. Izdelajte topološke metode za klasifikacijo enostavnih geometrijskih objektov. Osredotočite se na sfero in geometrijski torus ter odločite, kolikšen vzorec točk je potreben, da relativno zanesljivo odločimo, kateremu geometrijskemu objektu pripada.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Miha Polanc, z vpisno številko 63990299 sem avtor diplomskega dela z naslovom:

Koliko točk določa katero geometrijsko ploskev?

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Gašperja Fijavža,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 2. septembra 2016 Podpis avtorja:

Kazalo

Povzetek

Naslov: Koliko točk določa katero geometrijsko ploskev?

Geometrijsko ploskev lahko približno opišemo s končnim vzorcem njenih točk. V delu se ukvarjamo z vprašanjem, s koliko točkami, glede na način vzorčenja in rod ploskve, lahko zanesljivo rekonstruiramo originalno geometrijsko ploskev.

Najprej opišemo različne načine vzorčenja točk s ploskve, kaj je enakomerni in kaj je slučajni vzorec točk z izbrane ploskve. Vzorce lahko obravnavamo s topološkimi metodami, natančneje metodami vztrajne homologije. Iz vzorca točk s programskim paketom Javaplex konstruiramo filtracijo Vietoris-Ripsovih simplicialnih kompleksov in opazujemo črtni diagram Bettijevih števil.

V zaključku predstavimo računske rezultate za sfero in torus glede na različna modela vzorčenja, ter nekaj možnosti za nadaljnje izboljšave.

Ključne besede: Vietoris-Ripsov kompleks, Bettijeva števila, Javaplex, vztrajna homologija, sfera, torus.

Abstract

Title: How many points are needed to determine a geometric surface?

A geometric surface can be approximately described using a finite point-sample. The main question of this thesis is the following: how many points, depending on the sampling model and surface genus, are needed to confidently reconstruct the original geometric surface.

First we present different sampling models of surface points — the uniform and the random sample. We use topological methods, in particular persistent homology, to process our data. Using Javaplex software package we construct a filtration with Vietoris-Rips simplicial complexes and consider the bar-code diagram of its Betti numbers.

Finally we present our computational results for both the sphere and the geometric torus with respect to the two sampling models, and several options for further improvements.

Keywords: Vietoris-Rips complex, Betti numbers, Javaplex, persistent homology, sphere, torus.

Uvod

Ljudje imamo sposobnost, da iz dvodimenzionalne mreže raznobarvnih točk, ki jo zaznamo z očmi, sestavimo tridimenzionalno sliko okolice, v kateri razločimo posamezne objekte. Nekaj podobnega bi radi dosegli tudi z računalnikom. Torej bi za neko množico neodvisnih točk v prostoru radi vedeli, koliko objektom pripadajo, ter kakšne so njihove (topološke) značilnosti.

Gornji opis je malo posplošena razlaga, s čim se ukvarja to delo. Pri prepoznavi objektov si bomo pomagali z *vztrajno homologijo* [3]. Vztrajna homologija vključuje postopke za izračun topoloških značilnosti prostora pri različnih prostorskih resolucijah. Bolj vztrajne značilnosti so prisotne skozi večji razpon resolucij, zato smatramo, da predstavljajo dejanske značilnosti prostora, kot pa da so posledica napak pri zajemu podatkov, šumu oziroma določeni izbiri parametrov.

Vztrajna homologija je relativno nova metoda za izračun topoloških značilnosti naborov točk v prostoru. Prostor, v katerega so točke postavljene, je v resnici lahko poljubne dimenzije, ne samo 2- ali 3-dimenzionalen. Z značilnostmi mislimo na število neodvisnih *komponent*, število (in morda tudi velikost) zaprtih *prostorov*. Na splošno - število lukenj v različnih dimenzijah.

Podatki (točke) lahko predstavljajo različne stvari. Najbolj očiten primer vhodnih podatkov je množica točk v $\mathbb{R}^3$, ki je lahko pridobljena npr. z laserskim vzorčenjem objektov. Ni pa nujno, da podatki predstavljajo fizične objekte v prostoru. Pogost vir vhodnih podatkov so slike. Iz njih generiramo matriko velikosti $n \times 3$, kjer prvi dve koordinati predstavljata x in y koordinati slikovnega elementa, tretja koordinata pa je njegova svetlost.

Ali pa namesto svetlosti vzamemo kar RGB komponente posameznih točk in dobimo matriko $n \times 5$. Kar že lahko štejemo za objekt v višjih dimenzijah.

Z vizualizacijo objektov v višjih dimenzijah imamo ponavadi težave, vztrajna homologija pa nam lahko pri tem pomaga. Črtne kode lahko na enostaven način prikažejo nekatere lastnosti, ki jih sicer nebi mogli zaznati.

Ta naloga se ukvarja s precej poenostavljenim problemom in sicer razpoznavanjem dveh (vnaprej poznanih) tipov geometrijskih objektov. Podatke (množice točk za sfero in torus) pripravimo sami, s čimer se ukvarja prvi del naloge.

Ker vnaprej vemo, kakšne objekte lahko pričakujemo, smo pri razločevanju precej uspešni. Nimamo pa še pravila, kako prepoznati poljubne objekte na nepoznanih podatkih.

Pri izdelavi diplomske naloge sem uporabljal naslednjo programsko opremo:

- Javaplex 4.2.5
- MATLAB R2016a
- Microsoft Visual Studio Community 2015

Poglavje 1

Testni podatki

1.1 Vzorci

Za potrebe tega dela bomo zgradili naslednje tipe vzorcev n točk:

- enakomerno porazdeljene točke na sferi,
- slučajno porazdeljene točke na sferi,
- slučajno porazdeljene točke na sferi s perturbacijo, pri čemer vsako točko premaknemo za $\leq \overline{d}/3$, kjer parameter $\overline{d}$ predstavlja *idealno* razdaljo med točkama na enakomerno porazdeljeni sferi,
- slučajno porazdeljene točke na sferi s perturbacijo, pri čemer vsako točko premaknemo za $\leq d/3$, kjer parameter d predstavlja minimalno razdaljo med slučajno porazdeljenima točkama na sferi,
- enakomerno porazdeljene točke na torusu,
- slučajno porazdeljene točke na torusu.

Sfera je množica točk v prostoru, ki so enako oddaljene od središča/izhodišča, torus pa množica točk, enako oddaljenih od krožnice. Pri torusu torej nastopata dva radija; z velikim R bomo označili radij krožnice, z malim r pa razdaljo med krožnico in točkami.

Koliko točk potrebujemo za uspešno prepoznavo torusa, je odvisno od razmerja med radijema. Večje kot je, več točk bo potrebnih. Odločili smo se,

da bomo iskali najmanjše število točk na (pohlevnem) geometrijskem torusu z razmerjem radijev $R : r = 3 : 1$.

1.2 Enakomerna in slučajna razporeditev točk

Na tem mestu razložimo razliko med enakomerno in slučajno razporeditvijo točk na ploskvi.

Zamislimo si ploskev s 1000 točkami in na njem neko zaplato, ki predstavlja denimo 15% celotne površine objekta. Na takšni zaplati pričakujemo v povprečju 150 točk, torej premo sorazmerno glede na delež površine zaplate. To velja tako za enakomerno kot slučajno razporeditev.

Vendar, pri enakomerni razporeditvi se dejansko število ne bo prav bistveno razlikovalo od pričakovanega. Pri slučajni se lahko pa precej, in nič nenavadnega ni če bo dejansko število točk samo polovica pričakovanih, ali pa dvakrat toliko. Standardni odklon števila točk na zaplati bo torej večji.

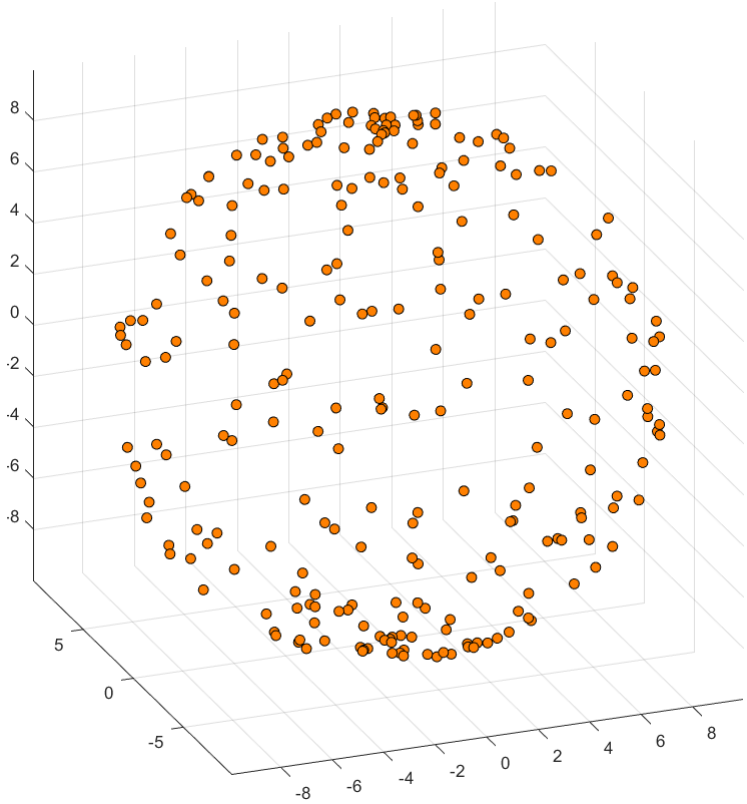
1.3 Sfera z enakomerno razporeditvijo

1.3.1 Začetna razporeditev

V prvem koraku s pomočjo naslednjih formul sestavimo začetno razporeditev točk na sferi. Z rnd označimo funkcijo, ki vrača naključno realno število med 0 in 1.
Posamezno točko za sfero kreiramo z algoritmom 1. Če algoritem 1 poženemo

Rezultat: x-, y-, in z-koordinata točke na sferi.
1 $\theta = \mathrm{rnd} \cdot 2 \cdot \pi$;
2 $\phi = \mathrm{rnd} \cdot 2 \cdot \pi$;
3 $x = R \cdot \cos(\theta) \cdot \cos(\phi)$;
4 $y = R \cdot \cos(\theta) \cdot \sin(\phi)$;
5 $z = R \cdot \sin(\theta)$;
Algoritem 1: Koda za začetno postavitev posamezne točke na sferi.

za 250 točk, pri radiju R=10, dobimo vzorec točk, prikazan na sliki 1.1.



Slika 1.1: Primer začetne postavitve točk na sferi.

Iz same slike je razvidno, da so točke bolj skoncentrirane na *polih*, kot pa na *ekvatorju*. V naslednjem koraku jih bomo enakomerno razporedili po ploskvi.

1.3.2 Razporejanje točk po sferi

Ko razmišljamo, kako bi dosegli enakomerno razporeditev, pridemo na idejo, da se morajo točke nekako odbijati med sabo. In sicer bližje kot so, močnejša je odbojna sila. Nekakšna anti-gravitacija torej.

Na vsako posamezno točko delujejo vse ostale točke, tako da jo hočejo premakniti v neko smer, za neko razdaljo. Sešteti moramo prispevke sil vseh preostalih točk in prestaviti opazovano točko glede na to vsoto. Seveda moramo poskrbeti, da točke ne uidejo s sfere, zato jih bomo po vsakem

premiku glede na silnice postavili nazaj na sfero. To ponavljamo, dokler se točke ne premikajo več znatno, ker so v numerični *ravnovesni* legi.

Rezultat: enakomerno razporejen vzorec točk na sferi
1 **while** *točke niso enakomerno razporejene* **do**
2 **foreach** *točka A na sferi* **do**
3 VektorOdbojnihSil = 0;
4 **foreach** *točka B na sferi* **do**
5 **if** *A!=B in sta si A in B dovolj blizu* **then**
6 VektorOdbojnihSil+=odbojna sila točke B na točko A;
7 **end**
8 **end**
9 prestavi točko A v smeri VektorOdbojnihSil;
10 projiciraj točko A nazaj na sfero;
11 **end**
12 **end**

Algoritem 2: Enakomerna razporeditev točk po sferi.

Zapišimo še nekaj komentarjev k algoritmu 2:

- Zanka v vrstici 2 teče po vseh točkah vzorca.
- Vektor odbojnih sil, ki deluje na točko A, v vrstici 3 nastavimo na 0 in
- v naslednjih štirih vrsticah prištevamo prispevke preostalih točk.
- V vrstici 9 točko A prestavimo glede na vektor odbojnih sil in jo
- v vrstici 10 projeciramo nazaj na sfero.

Pri algoritmu 2 bomo potrebovali še *algoritem za izračun odbojnih sil*. Če nas zanima odbojna sila točke B na točko A, je postopek naslednji:

$$\begin{aligned} F &= A - B \\ \|F\| &= \sqrt{x_F^2 + y_F^2 + z_F^2} \\ delež\, idealne\, razdalje &= \max\left[\frac{\|F\|}{idealna\, razdalja}, 0.1\right] \\ nova\, dolžina_F &= \frac{idealna\, razdalja}{(delež\, idealne\, razdalje)^p} \\ F &= nova\, dolžina_F \cdot \frac{F}{\|F\|} \end{aligned} \tag{1.1}$$

Pri tem upoštevamo, da sta točki A in B trirazsežna vektorja, z x-, y-, in z-koordinatami.

- Odbojno silo, ki jo v končni fazi znova označimo z F, s katero točka B deluje na točko A, izračunamo kot razteg njune razlike $A - B$.
- Dolžino razlike $A - B$ primerjamo s pričakovano končno razdaljo med sosednjimi točkami. Ker so lahko točke res zelo blizu skupaj (ali pa celo na istem mestu), nam to lahko povzroči probleme v naslednjem koraku.
- Za take primere imamo varovalko, ki razdaljo nastavi na vsaj 0.1.
- Izračunamo novo dolžino vektorja F, kjer je p neko število, s katerim potenciramo kvocient, p nastavljamo po želji. Večji kot bo, večji vpliv bodo imele bližnje točke, in manjši bolj oddaljene. Pri sferi uspemo že s $p = 1$, pri torusu (ki bo uporabil isti algoritem), pa ga bomo morali povečati.
- Nazadnje vektorju odbojnih sil F nastavimo novo dolžino.

Algoritem 2 ima ustavitveni pogoj, da razporejanje ponavlja, dokler točke niso enakomerno razporejene. Kako pa vemo, kdaj nadaljnje razporejanje ni več smiselno?

V ta namen si pomagamo s skupno *energijo objekta*, ki jo izračunamo na koncu vsake iteracije. V praksi je to seštevek dolžin odbojnih sil, ki delujejo med vsemi pari točk na objektu.
Če energija objekta med dvema iteracijama pade, potem vemo da moramo z razporejanjem nadaljevati. Če pa ostane ista, oz. je celo večja, pa to še ne pomeni da lahko že zaključimo z razporejanjem. Kar se v takem primeru dogaja je samo to, da sfera sicer je v grobem razporejena, nima več zgostitev na polih, ampak točke skačejo okoli svoje pozicije. Nekako jih je treba *umiriti*, in zmanjševati razdaljo, za katero se premaknejo pri razporejanju.

Zato v ta namen uvedemo še en kvocient, *MovementGranularity*. Na začetku iteriranja je nastavljen na 1, v primerih ko pa je nova energija objekta večja od prejšnje, ga nekoliko zmanjšamo, npr. pomnožimo z 0.95. Premik

točke vsakič pomnožimo z omenjenim kvocientom. Tako lahko računamo, da se bodo počasi ustalile na svojih mestih.

Ker preračunavanje odbojnih sil za čisto vse pare točk postane zelo potratno, ko je točk veliko, v izračunih upoštevamo samo pare točk, ki so paroma dovolj blizu, torej znotraj nekega radija. Radij je smiselno nastaviti tak, da bo pokril razdaljo vsaj do prvih nekaj sosedov na enakomerno razporejeni sferi. Empirično se je kot ustrezen izkazal šestkratnik idealne razdalje, kjer je *idealna razdalja* razdalja med sosednima točkama na enakomerno razporejeni sferi, in jo lahko dovolj dobro ocenimo z formulo (1.2):

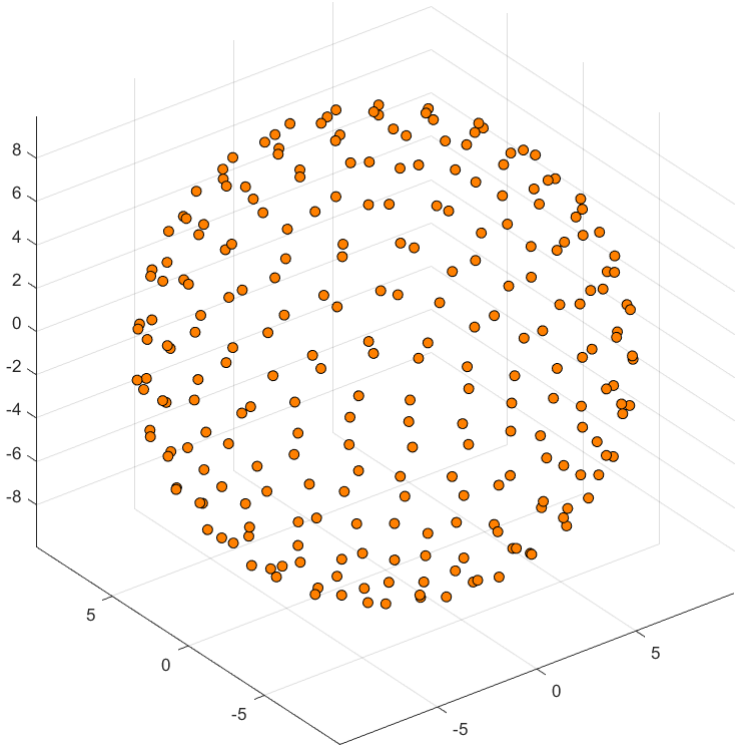
$$idealna\,razdalja = 2 \cdot \sqrt{\frac{površina\,na\,točko}{\pi}} \tag{1.2}$$

Koliko je površina na točko pa tudi ni težko uganiti — površina celotne sfere, deljeno s številom točk.

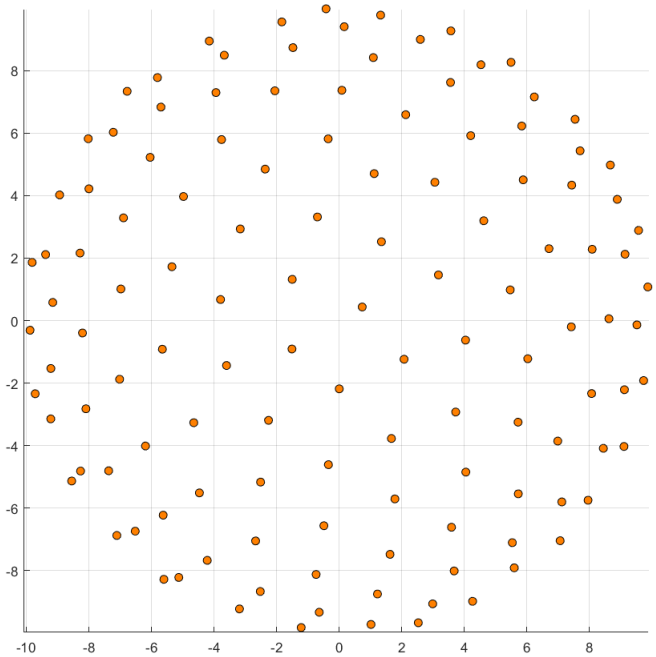
$$površina\,na\,točko = \frac{4 \cdot \pi \cdot R^2}{število\,točk} \tag{1.3}$$

Ko se postopek razporejanja izteče, je rezultat precej lepši od prvotne razporeditve. Rezultat je prikzan na sliki 1.2.

Na sliki 1.3 pa je zaradi preglednosti prikazana predstavljena samo polovica točk istega vzorca, ena sama hemisfera.



Slika 1.2: Enakomerno razporejene točke na sferi.



Slika 1.3: Polovica sfere z enakomerno razporeditvijo.

1.4 Sfera s slučajno razporeditvijo

Kreiranja vzorca slučajno razporejenih točk na sferi se lahko lotimo na več načinov. Obstaja nekaj algoritmov za generiranje točk [7].

Vzorce slučajno razporejenih n točk na sferi pa lahko na zadovoljiv način dobimo tudi s:

1. konstrukcijo vzorca N enakomerno razporejenih točk na sferi, kjer je $N \gg n$, in
2. slučajnim izbiranjem n izmed N točk omenjenega enakomernega vzorca.

Pri našem delu so zadoščali slučajni vzorci sfernih točk z največ 1000 točkami. Zato smo zgradili fiksen vzorec 20000 enakomerno razporejenih točk s sfere. Tako smo, denimo, slučajne vzorce s 45 točkami s sfere dobivali tako, da smo zaporedoma izbirali po 45 točk (brez ponavljanja) iz enakomernega vzorca z 20000 točkami.

Za slučajno razporejeno sfero smo naredili še dve variaciji, pri katerih vsako točko premaknemo za največ $d/3$ oziroma $\overline{d}/3$. Pri tem $\overline{d}$ predstavlja idealno pričakovano razdaljo med bližnjima točkama enakomernega vzorca, v drugem pa razdaljo med paroma točk, ki sta si v vzorcu najbliže.

1.5 Torus z enakomerno razporeditvijo

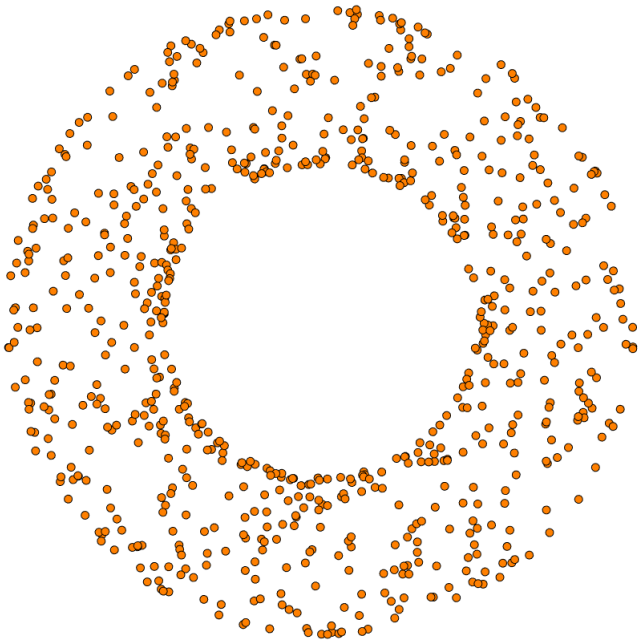
Torus T je množica točk, ki so od krožnice z radijem R (le-to imenujemo tudi nosilna krožnica torusa T) oddaljene za natanko r. Pri opisu torusa tipično privzamemo, da je $R > r$. Posledično R imenujemo *veliki radij (polmer)* torusa, r pa *mali radij* torusa T.

V primeru, ko je $R \gg r$, je torus T *zelo tanek*, po obliki podoben napihnjeni kolesarski zračnici. Torus, gledano od daleč, težko razločimo od krožnice. V nasprotnem primeru, ko je R le malo večji od r, si torus lahko predstavljamo kot kepo prebodeno z luknjo majhnega kalibra. Tudi na takšnem torusu iz grobe slike težko razpoznamo, ali je luknja dejansko

prisotna ali ne. V tem delu za torus privzamemo, da sta razmerji njegovih radijev opredeljeni z zvezo $R = 3 \cdot r$.

Algoritem 3 določi slučajno (a ne enakomerno izbrano) točko na torusu. Ko algoritem 3 poženemo za 750 točk, pričakujemo torusni vzorec, ki je

Rezultat: koordinate x, y, in z za posamezno točko na torusu
1 $\theta = \text{rnd} \cdot 2 \cdot \pi$;
2 $\phi = \text{rnd} \cdot 2 \cdot \pi$;
3 $x = R + r \cdot \cos(\phi) \cdot \cos(\theta)$;
4 $y = R + r \cdot \cos(\phi) \cdot \sin(\theta)$;
5 $z = r \cdot \sin(\phi)$;
Algoritem 3: Koda za začetno postavitev posamezne točke na torusu.



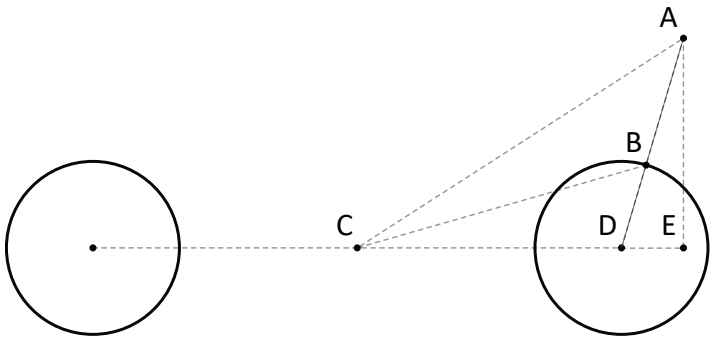
Slika 1.4: Primer začetne postavitve točk na torusu.

prikazan na sliki 1.4. Podobno kot pri sferičnem vzorcu točk, dobljenem z uporabo algoritma 1, so tudi tukaj točke porazdeljene neenakomerno. V bližini notranjega oboda so porazdeljene bolj pogosto kot v bližini zunanjega oboda.

Algoritem za izračun enakomernega torusnega vzorca točk je enak verziji za izračun enakomernega sferičnega vzorca, algoritmu 2, razlikuje se samo v treh podrobnostih:

- Površina torusa (ki jo potrebujemo za izračun idealne razdalje) se izračuna s formulo $4 \cdot \pi^2 \cdot R \cdot r$.
- Število p, s katerim potenciramo delež idealne razdalje, nastavimo na 3. Če bi, kot pri sferi, izbrali $p = 1$, končni rezultat ni najboljši.
- Projekcija premaknjene točke na torus je zahtevnejša operacija kot projekcija točke na sfero in jo opišemo v naslednjih vrsticah.

Denimo, da točka A ne leži na torusu. Želeli bi določiti projekcijo točke A na torus, ki jo označimo z B. Kot pomoč uporabimo sliko 1.5. Na sliki je prikazan pravokotni prerez torusa z ravnino Σ, ki poteka skozi središče torusa C in točko A. Presek ravnine Σ s torusom prikazujeta dve odebeljeni krožnici.



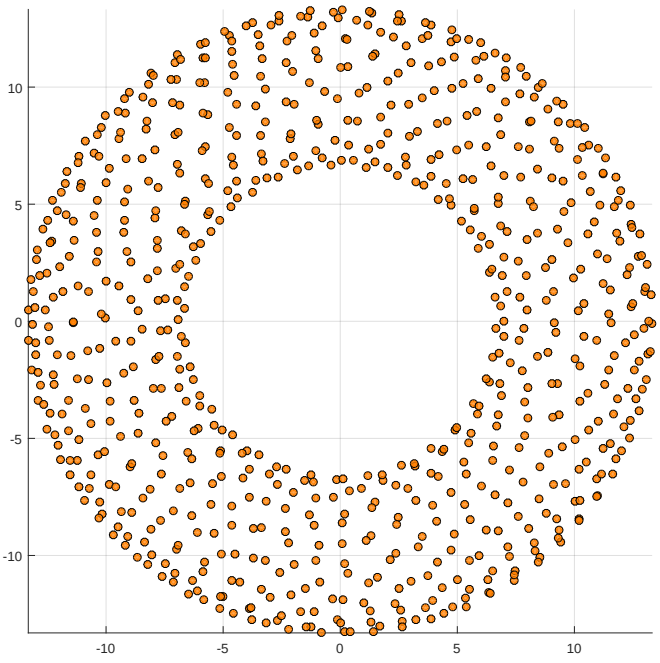
Slika 1.5: Projekcija točke A v točko B na torusu.

Preračun nam bosta olajšali dve smiselni predpostavki:

- osnovna krožnica torusa leži v ravnini xy, kar pomeni da imajo točke na krožnici koordinato $z = 0$.
- središče torusa (točka C) ima koordinate (0, 0, 0). Vsak vektor ki se začne v C, npr. CD, ima zato enake komponente kot so koordinate točke D.

Točko B, projekcijo A na torus, definiramo kot točko s torusa, ki je najbližje točki A. Določimo jo na naslednji način:

1. točka E je pravokotna projekcija točke A na ravnino nosilne krožnice torusa,
2. ravnina Σ je ravnina skozi središče torusa C in točki A in E,
3. točka D je točka na nosilni krožnici, ki leži na daljici CE, in nazadnje
4. točka B, projekcija točke A na torus, je presek daljice DE s torusom.



Slika 1.6: Enakomerno razporejen torus.

1.6 Torus s slučajno razporeditvijo

Slučajno razporejenega torusa se lotimo enako kot slučajno razporejene sfere. Imeti moramo pripravljen enakomerno razporejen torus z veliko točkami. Ocenili smo, da je 20000 točk dovolj.

Poglavje 2

Simplicialni kompleksi

Prepoznavanje objektov bo temeljilo na *simpleksih*, zato jih najprej definirajmo. Na tem mestu navajam citat, ki mi je najbolj razumljiv:

> (Simplex is) the most elementary geometric figure in Euclidean space of a given dimension. [4]

Torej najbolj enostaven geometrijski objekt, ki ga lahko zgradimo v izbrani dimenziji.

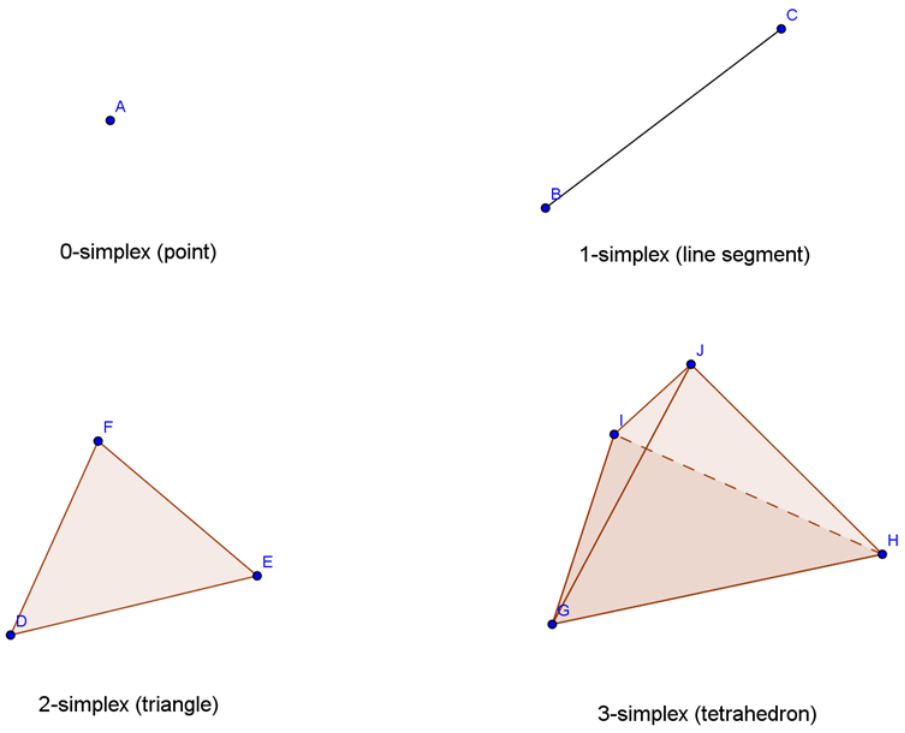
Za začetek v prostoru izberimo tri nekolinearne točke a, b in c — izkaže se, da tri točke vedno ležijo v isti ravnini. Točke a, b in c določajo trikotnik $\triangle abc$, tri stranice ab, ac in bc ter tri oglišča a, b in c. To strukturo, trikotnik, stranice, oglišča, kot celoto imenujemo *simpleks*, natančneje 2-*simpleks*.

Simpleks razsežnosti/dimenzije k, oziroma k-*simpleks* je formalno množica $k+1$ točk S, skupaj z vsemi njenimi podmnožicami. Vsako podmnožico $S' \subseteq S$, za katero je $|S'| = \ell + 1$, imenujemo ℓ-*lice* simpleksa S. Naš prvotni zgled, 2-simpleks na točkah a, b, c, vsebuje enorazsežna lica (daljice) ab, ac in bc, ničrazsežna lica a, b, c, pa tudi celoten trikotnik $\triangle abc$ kot 2-razsežno lice. Če je S k-simpleks in je $i \leq k$, potem je število njegovih i-razsežnih lic enako $\binom{k+1}{i+1}$. Vsako i-razsežno lice namreč pridelamo tako, da izberemo $i+1$ točk izmed $k+1$ točk simpleksa.

Za 3-simpleks, imenujemo ga tudi tetraeder, je število njegovih 2-lic (trikotnikov) enako $\binom{4}{3} = 4$, število robov/daljic, 1-lic, je enako $\binom{4}{2} = 6$, in

število njegovih oglišč, 0-lic, je enako $\binom{4}{1} = 4$. Glej tudi sliko 2.1.

Slika 2.1 prikazuje simplekse za prvih nekaj dimenzij. V dimenziji 0 je to točka, v dimenziji 1 daljica, v dimenziji 2 trikotnik, v dimenziji 3 tetraeder.



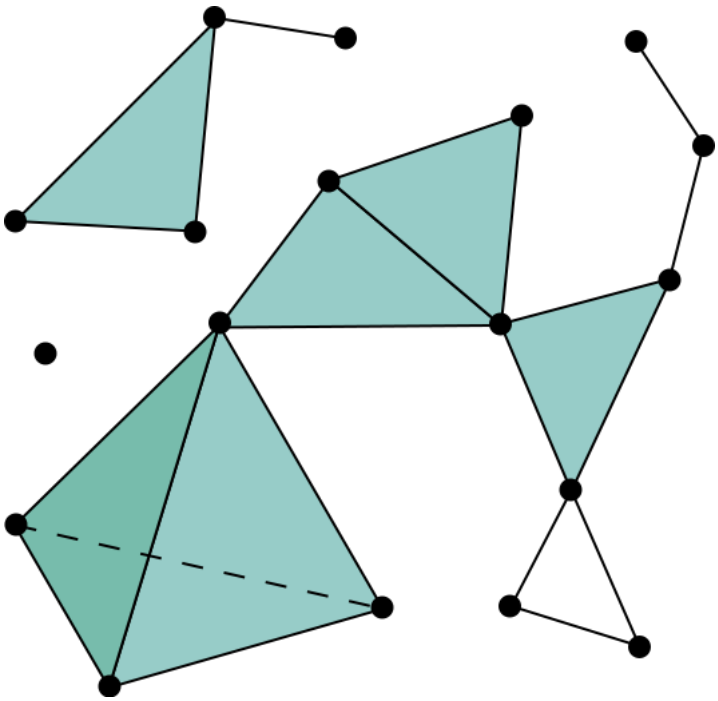
Slika 2.1: Simpleksi dimenzij med 0 in 3. Slika je vzeta iz [6].

Simpleksi se med seboj lahko lepijo v kompleksnejše strukture. *Simplicialni kompleks* K v $\mathbb{R}^n$ je unija simpleksov v $\mathbb{R}^n$, za katero velja:

(S1) lice vsakega simpleksa iz K je v K, in

(S2) presek dveh simpleksov v K je lice obeh simpleksov.

Dimenzija ali *razsežnost* simplicialnega kompleksa je najvišja dimenzija katerega izmed simpleksov, vsebovanih v njem. Kot primer, dimenzija kompleksa K_0, predstavljenega na sliki 2.2, je enaka 3.



Slika 2.2: Simplicialni kompleks K_0. Slika je vzeta iz [8].

.

2.1 Bettijeva števila

Struktura, ki nas bo zanimala pri posameznem objektu, so *Bettijeva števila*, $B_0, B_1, B_2, \ldots$, simplicialnega kompleksa K, ki opisujejo najosnovnejše topološke lastnosti. Bettijeva števila lahko v resnici definiramo tudi na splošnejših topoloških prostorih. Natančna definicija Bettijevih števil presega okvir tega dela. Bomo pa podali neformalne razlage za začetna Bettijeva števila B_0, B_1 in B_2.

Še beseda o sami oznaki: k-to Bettijevo število kompleksa K zapišemo kot $B_k(K)$; če pa je kompleks znan iz konteksta, bomo uporabljali tudi oznako B_k.

- B_0 je število povezanih komponent. Kompleks K_0 s slike 2.2 ima 3 komponente, zato je $B_0(K) = 3$.
- B_1 je število *enodimenzionalnih* lukenj. Z drugimi besedami, B_1 je število *tunelov* skozi objekt. Za simplicialni kompleks K_0 velja $B_1(K_0) =$

1, edina njegova enodimenzionalna luknja je *trikotnik* desno spodaj.

- B_2 je število *praznih prostorov*. Lahko si ga predstavljamo tudi kot največje število različnih barv, s katerimi bi lahko napolnili prostore objekta, ne da bi se barve mešale. Simplicialni kompleks K_0 je brez praznih prostorov, zanj velja $B_2(K_0) = 0$.

Zaporedje Bettijevih števil izbranega kompleksa je neskončno; za vsak objekt (ki ga lahko predstavimo kot končno unijo simpleksov) pa velja, da so od določenega mesta naprej vsa Bettijeva števila enaka 0. Vsekakor je $B_k(K) = 0$, kadar je k večji ali enak dimenziji kompleksa K.

Na sliki 2.3 lahko vidimo začetna Bettijeva števila B_0, B_1, B_2 in B_3 po vrsti za točko P, krožnico (1-sfero) S_1, sfero (2-sfero) S_2 in torus T.

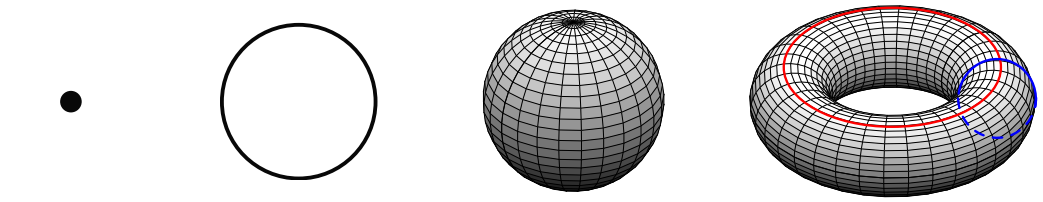
Število B_0 opisuje število povezanih komponent, vsi predstavljeni objekti, točka, krožnica, sfera in torus, so povezani. Zato je za vse omenjene objekte $B_0 = 1$.

Točka P je simplicialni kompleks (dejansko tudi simpleks) dimenzije 0. Zato za vsak $k \geq 1$ velja $B_k(P) = 0$. Krožnico S_1 lahko kot simplicialni kompleks zapišemo kot unijo treh daljic (1-simpleksov), njena dimenzija je enaka 1. Zato za vsak $k \geq 2$ velja $B_k(S_1) = 0$. Ima pa krožnica eno enodimenzionalno luknjo, zato je $B_1(S_1) = 1$.

Komentirajmo še Bettijeva števila za sfero S_2 in torus T. Oba objekta sta dvodimenzionalna, zato nas zanimajo zgolj še B_1 in B_2. Sfera enodimenzionalnih lukenj nima. Če na sfero narišemo enostavno sklenjeno krivuljo (ki domnevno opisuje luknjo), opazimo, da je njena notranjost zapolnjena. Torej velja $B_1(S_2) = 0$. Po drugi strani pa sfera ima dvodimenzionalno luknjo, loči namreč svojo notranjost od zunanjosti. Velja torej $B_2(S_2) = 1$.

Tudi torus T loči notranjost od zunanjosti, torej je $B_2(T) = 1$. Ima pa torus dve *različni* enorazsežni luknji. Na sliki 2.3 sta razvidni kot predstavnici razreda vzporednikov (rdeča krivulja) oziroma poldnevnikov (modra krivulja).

Iz slike 2.3 slutimo, da bomo objekte razločevali na podlagi Bettijevih



Slika 2.3: Prva štiri Bettijeva števila za točko, krožnico, sfero in torus.

števil. V naslednjem poglavju bomo opisali, kako priti do njih.

Poglavje 3

Vietoris-Ripsov kompleks

3.1 Uvod

V prejšnjem poglavju smo spoznali Bettijeva števila, v tem poglavju jih bomo poskušali izračunati. Pri tem naletimo na težavo: za izračun Bettijevih števil potrebujemo strukturiran simplicialni kompleks, imamo pa samo končno množico točk v prostoru. Točke so sicer opremljene s koordinatami, sicer pa so neodvisne ena od druge. Ne vemo na kakšen način (in če sploh) jih lahko povežemo med sabo.

3.2 Vietoris-Ripsovi kompleksi

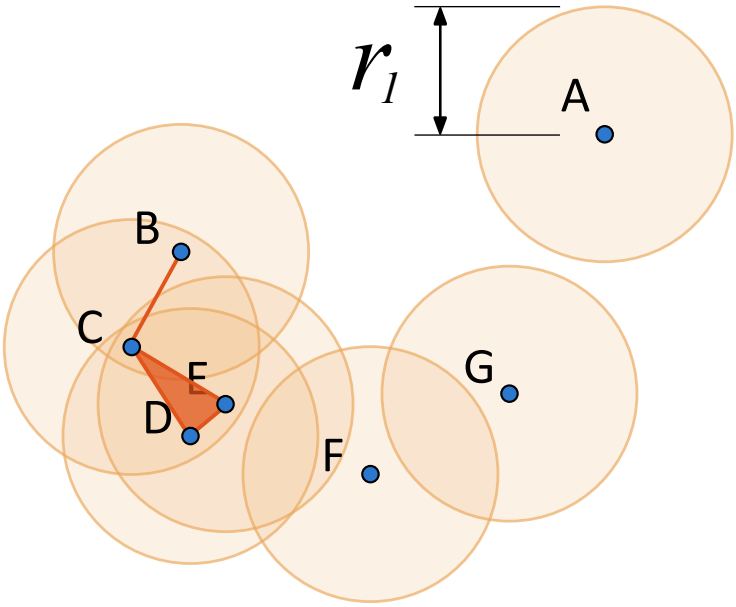
Izberimo realno število $r > 0$, recimo mu *resolucija*. *Vietoris-Ripsov kompleks*[1] ali krajše *VR-kompleks* — dimenzije d (v našem primeru je $d = 3$) za vzorec točk $\mathcal{E}$ pri resoluciji r — je simplicialni kompleks, ki ga označimo s $K_{r,d}$, in je opisan takole:

1. če je $x_1, x_2, \ldots, x_{k+1}$ $(k+1)$-terica točk iz $\mathcal{E}$, za katero je $k \leq d$, in za vsaki točki x_i, x_j $(1 \leq i \leq j \leq k+1)$ velja, da sta na razdalji $\leq r$, potem točke $x_1, \ldots, x_{k+1}$ sestavljajo k-simpleks kompleksa $K_{r,d}$.
2. $K_{r,d}$ ne vsebuje nobenih drugih simpleksov.

[1] V literaturi včasih srečamo tudi srednje kratko ime *Ripsov kompleks*.

V našem delu se omejimo na množice točk iz 3-dimenzionalnega prostora, v splošnem bi dopuščali tudi več dimenzij v podatkih.

Za primer poglejmo sliko 3.1, ki prikazuje množico točk $\{A, B, C, D, E, F, G\}$ in zgradimo VR-kompleks K_{r_1}. Hkrati s točkami je označena tudi resolucija r_1, okoli vsake izmed točk je narisana krožnica z radijem r_1. Simpleksi VR-kompleksa K_{r_1} so vse podmnožice točk, ki so na vzajemni razdalji $\leq r_1$. To pomeni, da K_{r_1} vsebuje, denimo, 1-simpleks/daljico BC. Prav tako so povezane točke C, D in E, med katerimi dobimo trikotnik/2-simpleks $\triangle CDE$. Dobljeni VR-kompleks K_{r_1} s slike 3.1 je dovolj enostaven, da lahko

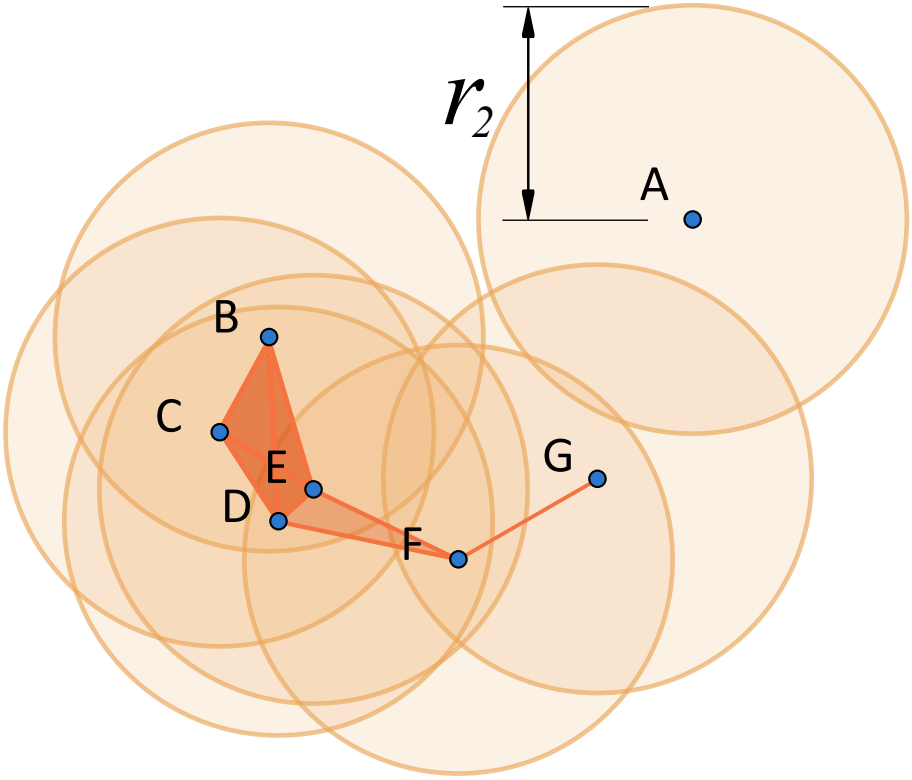


Slika 3.1: Ilustracija VR-kompleksa pri majhni resoluciji r_1.

sami določimo njegova Bettijeva števila. Velja $B_0(K_{r_1}) = 4$, $B_1(K_{r_1}) = 0$, $B_2(K_{r_1}) = 0$.

Postavi se vprašanje, kakšno resolucijo moramo izbrati, da dobimo Bettijeva števila, reprezentativna za objekt. Če izberemo drugačno resolucijo, se VR-kompleks lahko spremeni.

Na sliki 3.2 imamo predstavljeno isto množico točk kot na sliki 3.1, nova resolucija r_2 pa je večja. Opazimo lahko, da kompleks K_{r_2} vsebuje 3-simpleks na točkah B, C, D, E. Ravno tako K_{r_2} vsebuje nekaj dodatnih trikotnikov



Slika 3.2: Ilustracija Vietoris-Ripsovega kompleksa pri večji resoluciji r_2.

in daljic. Posledično so se spremenila Bettijeva števila. Velja $B_0(K_{r_2}) = 2$, $B_1(K_{r_2}) = 0$, $B_2(K_{r_2}) = 0$.

Če resolucijo povečamo na maksimalno razdaljo med točkama iz množice točk, je dobljeni VR-kompleks kar simpleks. Vse točke bodo povezane z vsemi v enem samem maksimalnem simpleksu, kar bo za nas prav tako neuporabno kot množica nepovezanih točk.

Ker ne poznamo *optimalne* resolucije, s katero bi dobili reprezentativen VR-kompleks, se bomo problema lotili drugače. Izračunali bomo VR-komplekse in pripadajoča Bettijeva števila pri več *različnih* resolucijah. Tiste značilnosti, ki se bodo obdržale dolgo časa, skozi različne resolucije, bomo smatrali za reprezentativne za objekt. Z značilnostmi imamo v mislih *luknje* različnih dimenzij, ki jih predstavljajo Bettijeva števila.

3.3 Javaplex

Javaplex [1] je programski paket napisan v Javi, ki med drugim omogoča izračune VR-kompleksov na vzorcih podatkov, ter pripadajočih Bettijevih števil. Uporabna vrednost Javaplexa je v tem, da zna zgraditi VR-komplekse za vrsto enakomerno naraščajočih resolucij. Za vsak VR-kompleks izračuna Bettijeva števila, ter rezultate, Bettijeva števila pri določenih resolucijah, tudi grafično prikaže.

Javaplex je primarno namenjen integraciji v Matlab, od koder kličemo njegove funkcije. Lahko pa ga uporabljamo tudi kot knjižnico v Javi. Navajam enostaven primer uporabe iz skripte *Javaplex tutorial*, ki je del paketa:

Primer 3.1: Enostaven primer izračuna VR-kompleksa in Bettijevih števil v Javaplexu.

```
%% House Example

max_dimension = 3;
max_filtration_value = 4;
num_divisions = 100;

% create the set of points
point_cloud = ...
    examples.PointCloudExamples.getHouseExample();

% create a Vietoris-Rips stream
stream = api.Plex4.createVietorisRipsStream(point_cloud, ...
    max_dimension, max_filtration_value, num_divisions);

% get persistence algorithm over Z/2Z
persistence = ...
    api.Plex4.getModularSimplicialAlgorithm(max_dimension, 2);

% compute the intervals
intervals = persistence.computeIntervals(stream);

```

```
19 % create the barcode plots
20 options.filename = 'RipsHouse';
21 options.max_filtration_value = max_filtration_value;
22 options.max_dimension = max_dimension - 1;
23 plot_barcodes(intervals, options);
```

Komentirajmo zgornji primer:

- Med vrsticami 3 in 5 nastavimo nekaj parametrov:

  max_dimension je največja dimenzija, do katere Javaplex gradi simplekse v simplicialnem kompleksu. V našem primeru

  $$max\_dimension = 3$$

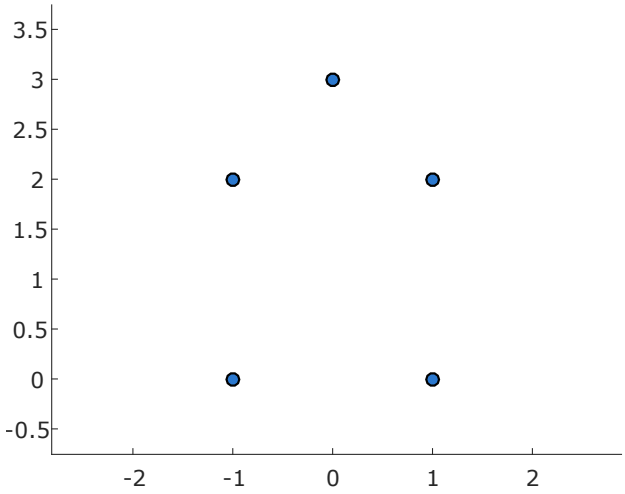
  pomeni, da kompleks lahko vsebuje simplekse s štirimi točkami, ne more pa vsebovati simpleksov s strogo več kot štirimi točkami. Največja možna dimenzija dobljenega kompleksa je torej enaka 3. Posledično bodo Bettijeva števila B_k, za $k \geq 3$, enaka 0.

  max_filtration_value je največja resolucija, do katere Javaplex izračunava VR-komplekse in Bettijeva števila. V našem primeru je enaka 4. To je več kot znaša *premer* vzorca točk — premer je maksimalna razdalja med točkama v vzorcu. Če bi dopuščali simplekse poljubnih dimenzij, bi pri največji resoluciji v kompleksu dobili en sam velik simpleks.

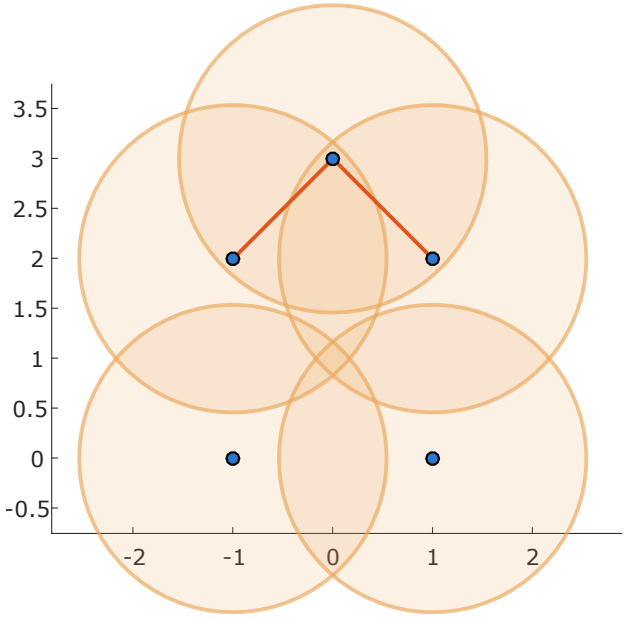
  num_divisions je število različnih vrednosti resolucije, pri katerih izračunamo Bettijeva števila, te vrednosti so členi aritemtičnega zaporedja z začetkom v 0 in koncem v max_filtration_value.

- Vrstica 8 kreira matriko z imenom *point_cloud* dimenzije 5×2, imamo namreč pet točk, od katerih ima vsaka po dve koordinati.
- Vrstice od 14 do 17 so jedro programa. Tukaj se dejansko izračunavajo VR-kompleksi in *luknje*, Bettijeva števila.
- Med vrsticami 20 in 23 samo še prikažemo graf s *črtnimi kodami* (slika 3.6),

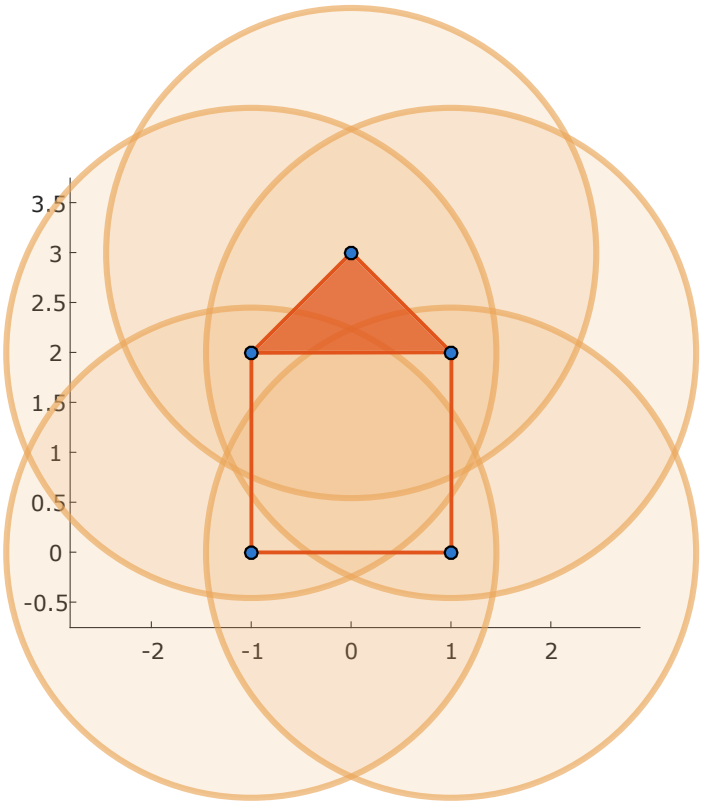
ki predstavljajo intervale resolucije, v katerih je obstajala posamezna luknja.



Slika 3.3: Primer vzorca točk, za katerega računamo Bettijeva števila.



Slika 3.4: Isti primer, pri resoluciji $r = 1.5$.

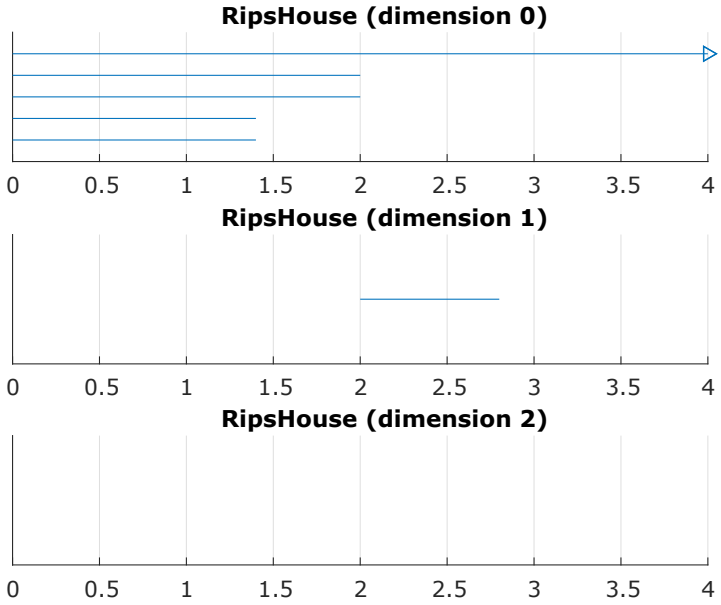


Slika 3.5: Isti primer, pri resoluciji $r = 2.5$.

Slika 3.3 prikazuje koordinate točk iz primera 3.1. Točki $(-1, 2)$ in $(0, 3)$ sta si najbližje, sta na razdalji $\sqrt{2}$. Vzajemno najbolj oddaljeni točki sta $(1, 0)$ in $(0, 3)$, ki sta na razdalji $\sqrt{10}$. Točk je 5, torej pričakujemo da bo Bettijevo število B_0 pri najmanjši resoluciji $r = 0$ enako 5. Pri $r = \sqrt{2}$ (ko se streha poveže v eno komponento) se B_0 zmanjša na 3 (slika 3.4). Pri $r = 2$ pričakujemo, da bo število komponent padlo na 1, hkrati se bo pojavila prva enodimenzionalna luknja (slika 3.5).

Na sliki 3.6 razberemo črtne kode filtracije vzorca točk s slike 3.3. Število B_0 iz začetne vrednosti 5 najprej pade na 3, nato pa od vrednosti resolucije $r = 2$ *vztraja* na 1. Število B_1 je pri začetnih resolucijah enako 0, nato pa pri resolucijah med 2 in (dejansko) $2\sqrt{2}$ vztraja na 1. Pri večji resoluciji je B_2 znova enako 0, saj luknjo v stavbi zapolnemo.

Iz črtnih kod razberemo da je resolucija, reprezentativna za objekt, med



Slika 3.6: Črtne kode, izračunane iz zaporedja Bettijevih števil na primeru iz slike 3.3.

2 in približno 2.8 (kar ustreza $2\sqrt{2}$). V tem intervalu so Bettijeva števila enaka $B_0 = 1, B_1 = 1$ in $B_2 = 0$. Če te številke primerjamo z referenčnimi vrednostmi na sliki 2.3, ugotovimo da imamo opravka z objektom, ki sodeč na luknje spominja na krožnico.

3.4 Kriteriji za razpoznavanje sfer in torusov

Vrnimo se zdaj k osnovnemu vprašanju naloge: *Koliko točk določa katero ploskev?* Iščemo torej najmanjše število točk N, ki še omogoča zanesljivo pravilno razpoznavo objekta. Najmanjše število potrebnih točk bomo iskali za vsak tip objekta posebej, ravno tako pa pri izbranem tipu tudi za vsak model vzorčenja. Pričakujemo lahko, da bodo števila potrebnih točk različna po tipih in modelih vzorčenja.

Za začetek bomo definirali kriterij, ki določa kdaj je N točk dovolj za pravilno razpoznavo objekta O pri izbranem modelu vzorčenja.

> Število N točk zadostuje za zanesljivo prepoznavo objekta O, ko izmed 1000 različnih razporeditev N točk po objektu O v izbranem modelu, v vsaj 95% dobimo Bettijeva števila, ki opisujejo objekt O.

Najmanjše potrebno število točk bomo iskali z bisekcijo. Za začetno spodnjo mejo, ki (zagotovo) ne zadošča za prepoznavo objekta bomo nastavili 1. Začetno zgornjo mejo (pri kateri dosežemo vsaj 95% uspešno prepoznavo) bomo poiskali s poizkušanjem. Začetno (domnevno) zgornjo mejo nastavimo pri $N = 100$ in, če ni dovolj, podvajamo število, dokler ne dosežemo vsaj 95% uspešnosti.

Ko imamo začetno spodnjo in zgornjo mejo N, z bisekcijo ožimo interval, dokler ne dobimo končnega najmanjšega števila točk, ki še zadošča za prepoznavo določenega objekta.

Postopek iskanja potrebnega števila točk ni čisto deterministične, testiramo namreč slučajne vzorce točk. Vseeno lahko pričakujemo, da je 1000 vzorcev fiksne velikosti dovolj, da slučajna izbira vzorcev ne vpliva na pravilnost naših rezultatov.

3.5 Avtomatizacija prepoznavanja reprezentativnih intervalov

Iz črtnih kod na sliki 3.6 smo sami po občutku razbrali interval, reprezentativen za objekt. Za avtomatsko klasifikacijo objektov pa potrebujemo dobro definiran kriterij, s katerim lahko računalnik sam določi reprezentativen interval.

V dostopni literaturi o uporabi vztrajne homologije nismo zasledili univerzalnega kriterija, ki bi omogočal avtomatski izračun Bettijevih števil iz diagrama črtnih kod. Kriterij, ki loči sfero od torusa, takšna objekta pa od preostalih, smo določili čisto empirično.

V ta namen potrebujemo podatke o črtnih kodah za nekaj različnih

sferičnih in torusnih vzorcev točk. Uporabimo jih, da določimo ustrezne kriterije. Pri tem naletimo na naslednjo težavo.

Denimo da zaženemo Javaplex na vzorcu 100 točk ter nastavimo dovolj veliko resolucijo, ki zagotavlja da je zadnji kompleks v filtraciji dejansko simpleks — maksimalna resolucija je večja kot premer vzorca točk. V tem primeru se zgodi eno od dvojega: ali program javi napako (premalo pomnilnika) in se ustavi ali pa preračunava nerazumno dolgo časa, dokler ga ne ustavimo sami. Razlog je v časovni in prostorski kompleksnosti iskanja vseh možnih kombinacij simpleksov.

Če je točk malo, denimo 30, teh težav ni.

Kaj lahko naredimo? Po eni strani lahko zmanjšamo število intervalov, za katere Javaplex izračunava VR-komplekse. Ampak s tem se ne izognemo izračunavanju vseh možnih simpleksov, ki jih dobimo pri največji resoluciji.

Bolj bi pomagalo, če bi lahko zmanjšali maksimalno resolucijo, ker bi s tem bistveno zmanjšali število različnih simpleksov. Če pogledamo npr. torus na sliki 1.4, opazimo da tako velike resolucije, ki bi pokrila celoten premer objekta, najbrž niti ne rabimo. Resolucija mora biti dovolj velika, da se vsaka posamezna točka poveže z sosednjimi, in se zgradi *mreža* trikotnikov, ki predstavlja sfero oz. torus. Kakšna je ta resolucija pa vnaprej ne vemo.

S preizkušanjem se je izkazalo, da je dovolj dobra rešitev, če nastavimo tako največjo resolucijo, da na koncu filtracije pridelamo kompleks, pri katerem je vsaka točka vsebovana v vsaj 30 daljicah kompleksa. S tako omejitvijo maksimalne resolucije smo pridelali uporabne rezultate, hkrati pa smo s to izbiro ostajali znotraj programskih zmožnosti.

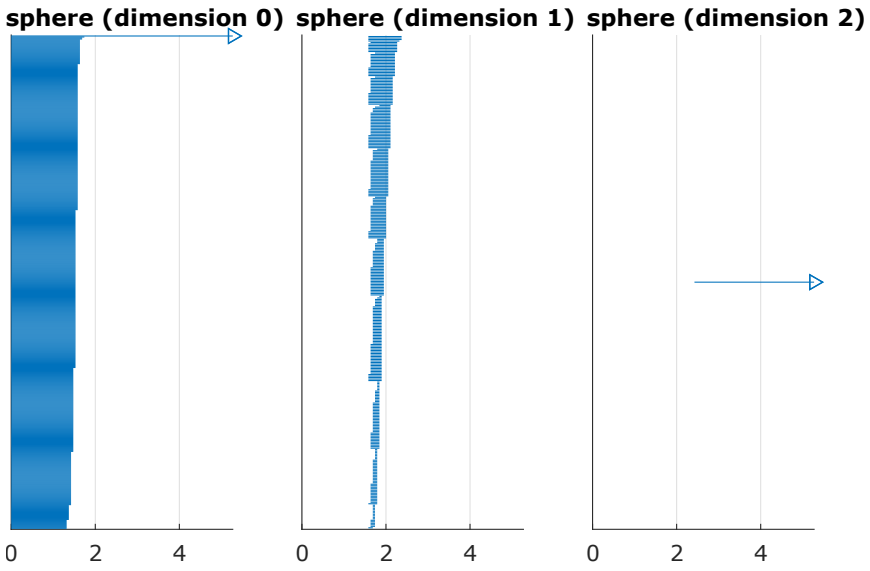
Poglejmo torej nekaj črtnih kod za različne objekte. Iz slik 3.7, 3.8, 3.9, in 3.10 razberemo naslednje pravilo za določanje reprezentativnega intervala:

1. izmed intervalov v dimenziji 2 bomo izbrali tiste, ki se končajo pri največji resoluciji, imenujmo jo R_2,
2. med slednjimi pa izberemo maksimalnega, tistega ki se začne pri najmanjši resoluciji, oziroma tistega/tiste, ki se začnejo najprej. To resolucijo označimo z r_2. Bettijevo število B_2 določimo kot število intervalov,

ki vsebujejo interval $[r_2, R_2]$.

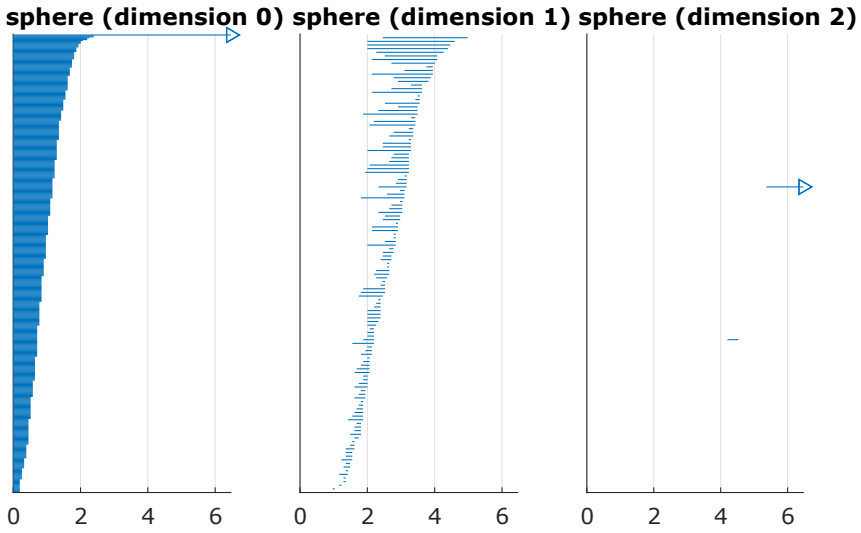
3. Če je $B_2 \neq 1$, potem izračun ustavimo. Predstavljeni objekt ni niti sfera niti torus. Tako za sfero kot torus namreč velja $B_2 = 1$.
4. Če je $B_2 = 1$, potem preštejemo intervale v dimenzijah 0 in 1, ki vsebujejo r_2. Ti predstavljajo B_0 in B_1.

Če z zgornjim pravilom ne dobimo Bettijevih števil, ki bi ustrezala ali sferi ali torusu, bomo uporabili rezervno strategijo — statistiko. Zanimale nas bodo samo dolžine intervalov (ℓ), ne pa tudi kdaj se pojavijo oz. izginejo. Intervale v vsaki dimenziji bomo uredili po dolžini v vrsto, nato pa izračunali prvi (Q_1) in tretji (Q_3) kvartil njihovih dolžin in interkvartilni raznik $IQ = Q_3 - Q_1$. Za izračun Bettijevih številih bomo upoštevali samo intervale zadostnih dolžin, tiste, za katere velja $\ell \geq Q_3 + 3 \cdot IQ$.

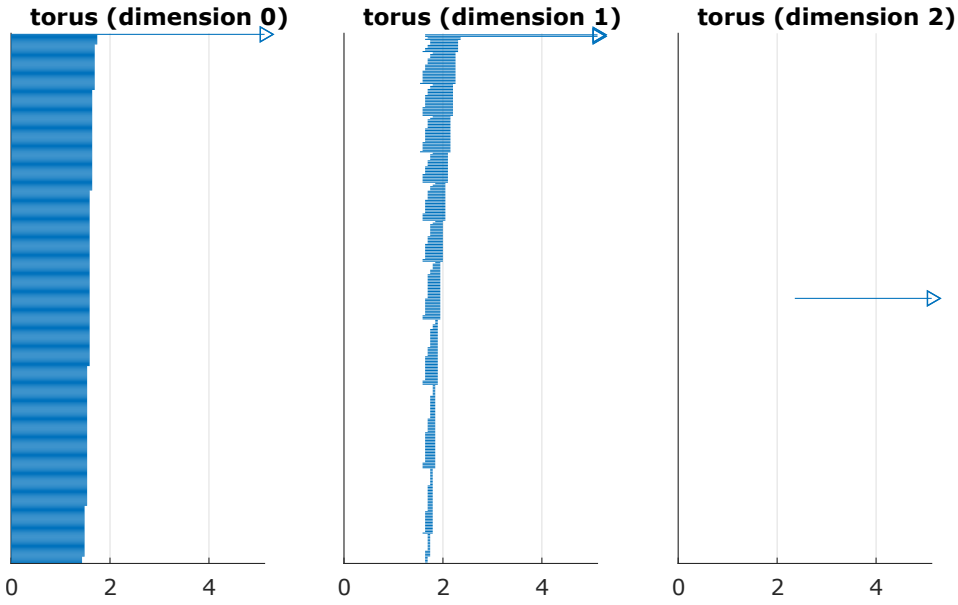


Slika 3.7: Črtne kode za 500 točk, enakomerno razporejenih po sferi.

Komentirajmo primer na sliki 3.8. Točke so slučajno razporejene na sferi z radijem 10. Največja resolucija, do katere računamo Bettijeva števila, je približno 6.5. To je namreč najmanjša resolucija, ki še zagotavlja, da je



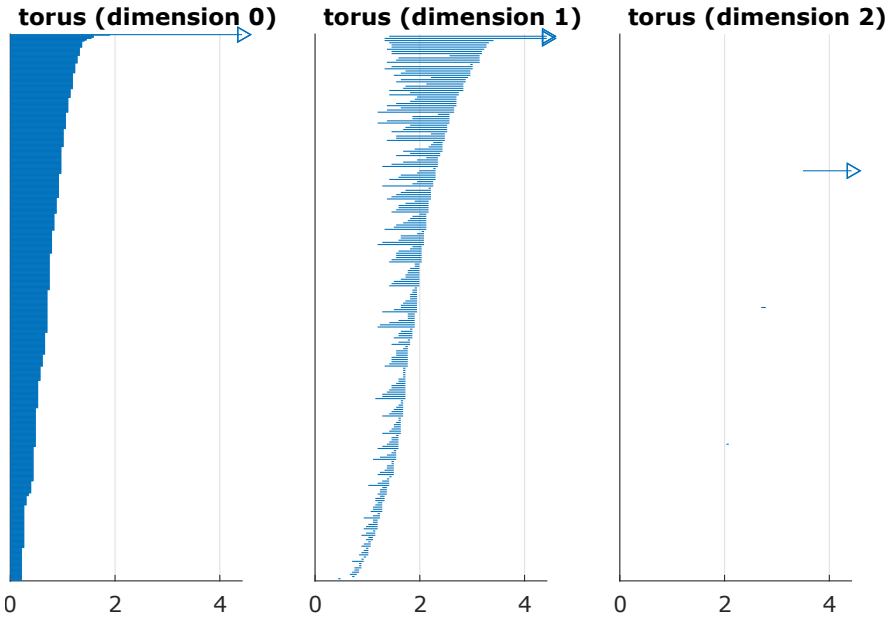
Slika 3.8: Črtne kode za 500 točk, slučajno razporejenih po sferi.



Slika 3.9: Črtne kode za 500 točk, enakomerno razporejenih po torusu.

vsaka točka vsebovana v vsaj 30 daljicah kompleksa. Intervali, ki so še vedno prisotni pri tej resoluciji, se končajo s puščico.

Ker je točk 500, se v dimenziji 0 pri najnižji resoluciji začne 500 intervalov. Ko se resolucija veča, se izolirane točke začnejo povezovati v večje komponente kompleksa. Vsakič, ko se dve komponenti združita v eno, se



Slika 3.10: Črtne kode za 1000 točk, slučajno razporejenih po torusu.

število intervalov zmanjša za 1. Tako pri resoluciji približno 2.4 ostane samo še ena komponenta — interval, ki vztraja do največje resolucije.

Intervali v dimenziji 1 so bolj zanimivi. Spomnimo se neformalne razlage za Bettijevo število B_1: število tunelov skozi objekt. Z večanjem resolucije se po sferi začne plesti *mreža* kompleksa, ki pa ima v površini še polno lukenj. Te luknje delujejo kot prej omenjeni tuneli. Ko resolucija naraste čez premer posamezne luknje se luknja zapre, posledično se interval zaključi. Iz intervala, ki se zaključi nazadnje, lahko sklepamo da je imelo na sferi največje področje brez točk premer približno 5.

V dimenziji 2 sta dve luknji. Krajšega, ki se začne in tudi zaključi malo čez 4, ne pričakujemo, sfera ima vendar samo 1 prostor. Razlog je v tem, da lahko pri določeni resoluciji in postavitvi točk dobimo domneven *prazen prostor*, tudi če ga dejansko ni. Kot primer tega pojava si bralec lahko predstavlja oglišča pravilnega šestkotnika s premerom denimo 5, ter nariše VR-kompleks z resolucijo malo manj kot 5. Rezultat bo VR-kompleks, ki bo ustrezal stranicam oktaedra — za katerega velja $B_2 = 1$. Drug, daljši interval seveda predstavlja prostor v sferi. Ker prostor v sferi lahko zaznamo šele, ko

je sfera brez lukenj, se morajo najprej zaključiti vsi intervali v dimenziji 1. Ker pa z B_1 ne zaznamo zadnje luknje v sferi, se mora zapreti tudi ta.

Poglavje 4

Rezultati

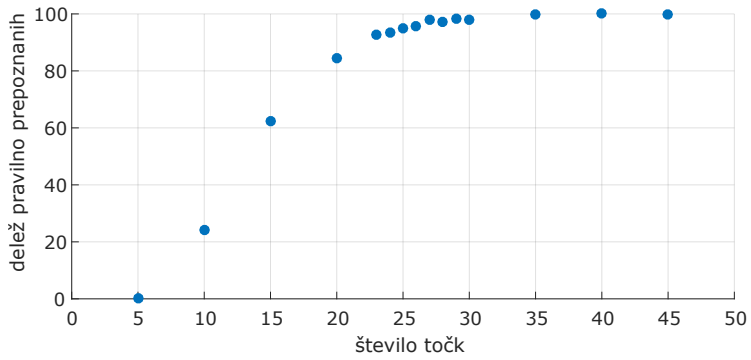
Zdaj ko poznamo postopek za izračun, lahko poženemo izračun za tipe objektov, naštetih v uvodu. Rezultati so zbrani v spodnji tabeli.

objekt	**min(n)**
sfera, enakomerna porazdelitev	6
sfera, slučajna porazdelitev	26
sfera, slučajna porazdelitev s perturbacijo $\leq d/3$	25
sfera, slučajna porazdelitev s perturbacijo $\leq \overline{d}/3$	38
torus, enakomerna porazdelitev	300
torus, slučajna porazdelitev	475

Slika 4.1 prikazuje delež uspešnih prepoznav glede na število točk; na primeru slučajno razporejene sfere. Za števila točk blizu 25 so deleži izračunani bolj pogosto.

Iz rezultatov lahko naredimo nekaj zaključkov. Torus je očitno precej bolj zahteven za prepoznavo kot sfera. To smo lahko pričakovali, za prepoznavo podrobnosti (v našem primeru volumen torusa) potrebujemo precej bolj gosto posejane točke.

Perturbacija sfere vpliva na rezultate, ampak samo če se točke premaknejo znatno.



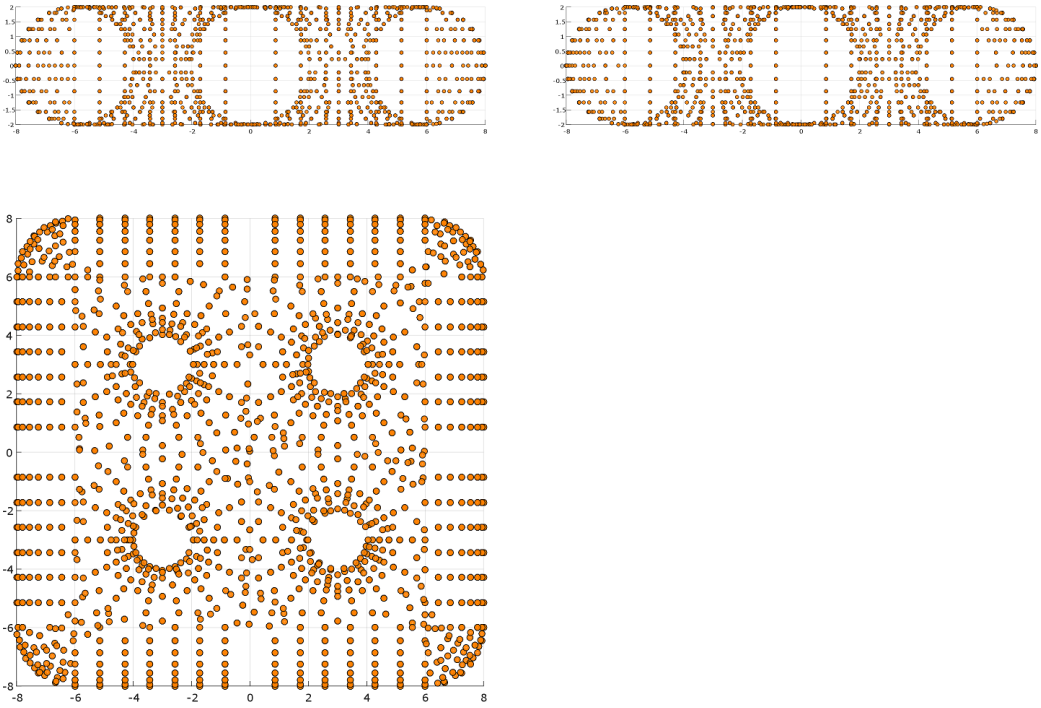
Slika 4.1: Delež uspešnih prepoznav na primeru slučajno razporejene sfere.

Najbolj nepričakovan rezultat je najbrž ta, da smo za prepoznavo malenkostno perturbirane slučajno razporejene sfere rabili manj točk kot pri enako razporejeni sferi brez perturbacije. Glede na to, da se rezultat ne razlikuje bistveno, gre verjetno samo za slučaj. Spomnimo se, da je meja za uspešno prepoznavo 950 primerov od 1000. Pri slučajno porazdeljeni sferi brez perturbacije, je bilo število uspešnih prepoznav malo manj kot 950. Pri veliko večji množici primerov bi se trend verjetno obrnil.

Verjetno nas zanima, kako se algoritma iz razdelka 3.5 obneseta na vzorcih točk, ki predstavljajo še kakšne drugačne objekte. Preizkusili smo jih torej še na ploskvi roda 4, ki je v več pogledih prikazana na sliki 4.2. Bettijeva števila, ki jih izračunamo s postopkom, opisanim v prejšnjem poglavju, so $B_0 = 1, B_1 = 4$ in $B_2 = 1$.

Interpretacija črtnih kod pri sferi in torusu je relativno enostavna, ne velja pa to za vse točkovne vzorce. Dovolj zanesljivo avtomatsko prepoznavanje Bettijevih števil za vsako vrsto vhodnih podatkov je po mojem mnenju izjemno težko doseči, če ne poznamo kakšne dodatne strukture vhodnega objekta.

Potrebna je pazljivost pri nastavljanju parametrov. Z večanjem resolucije zelo hitro presežemo kapaciteto pomnilnika, razen tega je čas izvajanja lahko zelo dolg. S premajhno resolucijo pa spet morda ne ujamemo nekaterih značilnosti v podatkih. Veliki nabori podatkov povzročajo težave in smo prisiljeni uporabit enega ali več *ovinkov*, da sploh pridemo do rezultatov.



Slika 4.2: Vzorec točk s ploskve roda 4, trije različni pogledi.

Možnosti za izboljšanje prepoznavanja Bettijevih števil so precejšnje. Kar pa presega namen te naloge, kjer smo iskali čim boljše pravilo za avtomatski izračun Bettijevih števil, s katerim smo iskali najmanjše število točk, ki predstavljajo sfero oz. torus. Vsekakor pa bi si s hitrejšimi izračuni lahko privoščili več eksperimentiranja in s tem verjetno prišli do boljšega pravila za prepoznavanje.

Če hočemo *pospešiti* izračune, je precej smiseln način predhodna priprava manjšega *podvzorca* točk, za katere nato izračunamo VR-kompleks. Načina izbiranja sta dva: naključno, ali pa gradnja vzorca, kjer samo prvo točko izberemo naključno, nato pa vse naslednje tako, da so čim bolj oddaljene od točk, ki so že v izboru. S tem *zredčimo* lokalne zgostitve, od katerih ne dobimo kakšne nove informacije, nam pa veliko prispevajo k nepotrebnemu računanju.

Načinov je še nekaj, kot zelo primernega se omenja uporaba *alfa kompleksov* [5] namesto VR-kompleksov.

Če pa hočemo *izboljšati* določanje Bettijevih števil, pa smo verjetno še najbolj prepuščeni lastni iznajdljivosti. Če vemo, kaj lahko pričakujemo v podatkih (kot smo v tej nalogi), potem lahko prilagodimo pogoje, kdaj nek interval predstavlja značilnost, in kdaj ne. Brez tega pa nam verjetno ne preostane drugega, kot da ročno pregledujemo diagrame črtnih kod.

Literatura

[1] H. Adams, M. Vejdemo-Johansson, A. Tauzs. "*JavaPlex: A research software package for persistent homology*". 2014. [Online] Dosegljivo: `http://appliedtopology.github.io/javaplex/` [Dostopano 18.8.2016].

[2] E. W. Chambers, V. de Silva, R. Ghrist. "*Vietoris-Rips complexes of planar point sets*". [Online]. Dosegljivo: `http://jeffe.cs.illinois.edu/pubs/rips.html` [Dostopano 21.8.2016].

[3] D. Choudhary, S. Bansal. "*Topological Data Analysis*", 2014. [Online]. Dosegljivo: `http://www.cse.iitk.ac.in/users/cs365/2014/_submissions/deepakc/project/report.pdf` [Dostopano 1.9.2016].

[4] Collins Dictionary. "*Simplex*". [Online]. Dosegljivo: `http://www.collinsdictionary.com/dictionary/english/simplex` [Dostopano 13.8.2016].

[5] R. Hennigan. "*A fast simplicial complex construction for computing the persistent homology of very large and high dimensional data sets*". [Online]. Dosegljivo: `http://www.cs.uml.edu/~rhenniga/main2.pdf` [Dostopano 30.8.2016].

[6] R. Nourai. "*Math Primer Series: Vectors III: Affine Spaces, Linear and Affine Combinations*". [Online]. Dosegljivo: `http://blogs.msdn.microsoft.com/rezanour/2011/06/26/math-primer-series-vectors-iii-affine-spaces-linear-and-affine-combinations/` [Dostopano 13.8.2016].

[7] E. W. Weisstein. "*Sphere Point Picking*". [Online]. Dosegljivo: `http://mathworld.wolfram.com/SpherePointPicking.html` [Dostopano 9.8.2016].

[8] Wikipedia. "*Simplicial complex*". [Online]. Dosegljivo: `http://en.wikipedia.org/wiki/Simplicial_complex` [Dostopano 13.8.2016].